blackduck ™

Best Practices for

Managing Software Intellectual Property

Executive Overview

Over the past ten years, new technologies like the Internet and open source software have enabled developers to fundamentally change the way they create software. Increasingly, distributed teams are collaborating to assemble software from reusable components and their own proprietary code rather than building applications entirely from scratch.

Spurred by ubiquitous connectivity, developers today can easily tap into rich software resources inside their own organizations and from around the world. They obtain software modules and libraries – and even code fragments – from enormous stores of high-quality, re-useable software components. Open source projects are a major new source of re-usable code, but so too are the growing in-house software repositories, and the increasing amounts of modular code from third-parties such as outsourcers.

With such broad availability, developers now focus on selecting the best available components for their projects, and rapidly incorporating them into their own intellectual property (IP) to deliver applications that provide optimal functionality, performance, and reliability.

The component-based development model is fundamentally changing the software industry. It enables organizations that develop software, either for commercial sale or for in-house use, to accelerate project timelines, improve software quality, and reduce development costs. But if not managed properly, the complexity inherent in this new world

Component-based development is **fundamentally changing** the software industry…

of 'mixed-IP' can pose business and technical risks to an organization.

This paper draws on the experiences of the Black Duck Software team, our customers, and other industry experts to propose new approaches to managing intellectual property in this new world. It describes a set of Best Practices that companies can use to avoid the risks and gain the benefits of this promising new approach to software development.

Component-based Development

A software application's design, implementation, and maintenance require the investment of precious development personnel, resources, and time. Development organizations have long understood the virtues of building new applications by re-using components already built and tested. Indeed, the evolution and rapid adoption of component-based architectures has been driven in part by their effectiveness in promoting economically-significant re-use. This effectiveness has stimulated the creation of commercial component libraries, which give development teams the option of purchasing pre-built components rather than acquiring the expertise and/or expending the time required to independently create them. In short, the approach can result in faster, less-expensive, and more effective software development.

It is therefore appropriate to consider components as intellectual property assets, optimize their usage, and protect their integrity. This applies equally to all components, whether they are internally developed or developed externally by a third party.

A development team intent on exploiting an externally developed commercial component does not typically purchase ownership of that component. Instead, they acquire a license to use that component in a specified manner – perhaps for only a certain number of developers working on a particular project, or with a specified royalty paid for each instance of a shipped product that includes the component.

Thus business judgment is required to ensure that the cost of licensing a commercial component is more than offset by the benefits – the classic make vs. buy tradeoff. Included in the cost analysis must be the effort required to ensure compliance with the license, e.g. limiting the component's usage to a specific project, or tracking shipments in order to accurately calculate royalty payments.

The Open Source Impact

Over the past five years, a powerful new approach to development – open source software – has risen to prominence, dramatically increasing the opportunity to re-use existing software. With today's powerful Internet search capabilities, developers can readily locate potentially useful components from among a wide array of re-usable software components. Re-use can take many forms, including bundling independent components, integrating with or using libraries, and incorporating source code or source code fragments. In some cases these components can be modified as required to improve functionality, quality, performance, or footprint. In many organizations, a developer's skill with Google and SourceForge.net is as important as

Faster, less expensive, and more effective software development…

his or her knowledge of software architecture and implementation.

As with commercial components, the ownership of externally developed open source components and fragments remains with their authors. While most of these authors allow the commercial use of their software without initial payments or royalties, many have chosen to impose other constraints, such as

- Attribution

- Usage reporting

- Publication of modifications and improvements

- License replication

- Resulting software must be open sourc

Such constraints are imposed by means of licenses, examples of which include

- the Apache Software License (ASL)

- the Common Public License (CPL)

- the GNU General Public License (GPL)

- the Mozilla Public License (MPL)

- the New BSD License

The Linux operating system, for example, is licensed under the GPL. A more complete listing of open source licenses is provided at http://www.opensource.org/licenses.

Open Source License Compliance

Development teams that incorporate open source components or fragments of open source components in their projects must comply with the terms of the

licenses associated with those components. This can be challenging for several reasons including the following.

- There is wide variation in the obligations imposed by open source licenses, ranging from the BSD license (which has few obligations) to the GPL license (which has many).

- Some open source licenses are legally complex, introducing constraints whose business implications may not be obvious to a developer choosing to re-use a component.

- The licenses of some commercial and open source components are mutually incompatible.

- The origin of a particular source code fragment may be difficult to determine, effectively obscuring its license obligations.

- Discovering the need to comply with a license late in a project's lifecycle can produce disagreeable tradeoffs, e.g. publishing all of the project's source code vs. increasing time-to-market by months while a component is replaced.

Open source and component reuse are here to stay…

In addition to the legal obligations imposed by licenses, developers who incorporate open source components into an organization's projects may either be obligated by the terms of the license or feel a moral obligation to give something back to the community. The resulting actions may result in the inadvertent dilution or loss of the organization's intellectual property (IP).

Management Alternatives

Organizations can react to the challenges of open source software licenses in one of three ways. Some organizations turn a blind eye, ignoring the issue until a catalyzing event or crisis occurs. But the resulting misfortunes – major code rewrites, embarrassing negative publicity, delayed sales, failed acquisitions – make this an increasingly untenable approach. This is especially true in the new environment of increased business transparency, executive responsibility, and potential shareholder lawsuits.

Other organizations take the Draconian approach of banning all open source software re-use. This strategy is flawed because it:

- Is difficult to enforce

- Decreases productivity and agility compared to organizations that successfully re-use externally developed components

- De-motivates development teams by requiring them to apply scarce resources to wheel re-invention rather than to moving forward

Further, both of the above approaches are ineffective because they fail to recognize the reality that open source and component reuse are here to stay.

The third and recommended alternative is to encourage the re-use of both internally developed and externally developed (commercial and open source) components, while establishing controls at critical points in the project lifecycle, for example:

- when components are first added to a project

- when internally developed components are created or modified

- at every build

- at each phase transition in the development process

- when considering the contribution of a component to an open source project or the transfer of its ownership to another party
- before acquiring a significant ownership interest in a software component

It is important to note that identifying problematic licensing issues early in the development cycle is tantamount to detecting serious software defects: the earlier a problem is detected, the less expensive it is to fix. While IP controls late in the development cycle – during QA or release assembly, for example – are better than none, the earlier they happen in the lifecycle, the better.

In the remainder of this report, we will outline several Best Practices that facilitate the management of software IP in the modern development organization. While all of these Best Practices encourage a focus on license compliance throughout the lifecycle, each organization should adopt the subset of Best Practices that meet their business needs. For example, an organization may wish to give its developers the flexibility to build prototypes using any open source code with no pre-approval, but specify a development phase transition beyond which all externally developed components must be approved. This provides the benefits of speed and efficiency to the developer, with the protection offered by a formal review.

It should be noted that, while all of these processes can be built and executed manually, their adoption and usage will be more effective and efficient when supported with an automated software compliance management system.

The **earlier** a problem is detected, the **less expensive** it is to fix...

Preparing for IP Management

When beginning to adopt Best Practices, several 'getting started' tasks should be considered by the individual or teams responsible for development and licensing. An organization intent on managing its software IP should identify the responsible business, legal, and technical individuals who will be involved in the process. The organization also should designate them as authorizers for each active project, and commission them as a group to oversee and manage the planning, implementation, and ongoing management of the process.

A good first step for the team is to define the boundary between internally developed and externally developed components (e.g. for business units, contractors, outsourcing organizations). For example, a small organization that prefers taking a conservative approach may deem all of the code developed within its walls to be "internal". Conversely, that company would view as "external" all software brought in from any outside source (e.g. licensed proprietary, open source, contractors' work product, etc.).
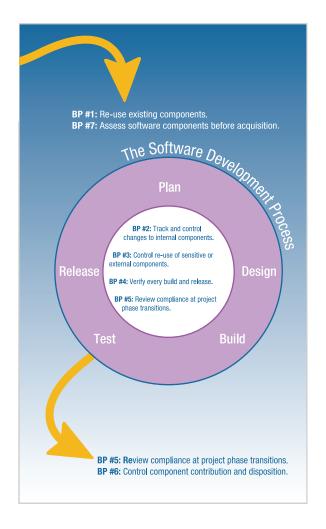
A more trusting organization might extend its view of "internal" software to include licensed proprietary software and software developed by its contractor and outsource partners. On the other hand, a department of a large corporation may want to consider its department "internal" and consider everyone else, including other departments in the same company to be "external." Business judgment must be used to determine where the boundary should lie.

Another key task in the preparation process is for the team to identify the development process phase transitions at which component re-use reviews will be conducted. The team also should define criteria for designating an internally developed component as sensitive (due to intellectual property value for example embodiment of trade secrets or patents, or risk), and develop and maintain a list of these sensitive items.

The organization also should consider establishing and maintaining lists of:

- Licenses that are prohibited by the organization

- Externally developed components that, based on previous reviews, are approved for use in projects, and the situations in which use is approved

- Internally developed components that, based on previous reviews, are approved for contribution to open source projects or disposition to third parties

Once these lists have been created, the organization can use them to conduct an initial assessment of its existing code base(s). In this important preparatory step, the organization identifies and establishes the baseline pedigree, licenses, and components in use. As with any process improvement, an acceptable alternative approach involves introducing these steps incrementally over time.

BP #1: Re-use existing components.
BP #7: Assess software components before acquisition.

The Software Development Process

Plan

BP #2: Track and control changes to internal components.

BP #3: Control re-use of sensitive or external components.

Release   Design

BP #4: Verify every build and release.

BP #5: Review compliance at project phase transitions.

Test   Build

BP #5: Review compliance at project phase transitions.
BP #6: Control component contribution and disposition.

Best Practice #1:
Re-use existing components wherever appropriate

Whenever a development team considers adding new features or refining existing functionality in a project, it should explicitly seek internally developed and externally developed components that could accelerate delivery. The team should establish criteria for selecting and procuring these components. As would be expected, any component under consideration that fails to meet functionality, performance, reliability, maturity, or risk requirements should be eliminated.

The team should also eliminate any externally developed component whose license is on the prohibited license list, whose license obligations are financially or legally incompatible with the project's business objectives, or that uses other external components or fragments whose licenses are similarly unacceptable. For example, an organization developing a product that will be delivered under a proprietary license needs to be certain that any open source or proprietary licensed code that is incorporated can be safely included without causing an irreconcilable conflict between licenses. Any components that pass this initial test should be subjected to a make-buy analysis to determine whether or not its acquisition makes sense from a business perspective.

Best Practice #2:
Track and control changes to internal components

To protect the organization's critical intellectual property, the creation and modification of all internally developed components should be tracked by recording a timestamp, the names of each author, and the applicable objectives and constraints. If a newly-created or modified component is suspected to be sensitive (e.g. a patent is sought, the code embodies algorithms that give the company significant competitive advantage, etc.), the project's legal, business, and technical reviewers should be convened. If these reviewers deem the component to be sensitive, they should add it to the organization's list of sensitive internal components.

Best Practice #3:
Control re-use of sensitive or external components

By assessing sensitivity and license obligations at the point where a component is first being considered for re-use, decisions can be based on verifiable facts, eliminating last-minute surprises, guesswork, compromises, and risk-taking. This dramatically reduces the risk of schedule slippage, cost overruns, and damage to the organization's reputation. It also helps prevent the inappropriate re-use of critical intellectual property.

For each component that a project's development team proposes to use within a project, the team should understand:

- The intended use and rationale for inclusion

- The component's sensitivity

- How the code will be incorporated

How the team deals with the component will depend, in part, on whether plans call for that component to be used temporarily or permanently. For example, the intent may be to use that component for a limited amount of time only to speed up prototyping or to advance the early phases of the development cycle, but not be intended to be made a permanent part of the code base.

Another determination the team should make is whether a component will be used only as is, or if modifications will be allowed, and if so, under which approvals.

Development teams should describe the method of joining that will be used to incorporate the component into the project. This is an effective step because different types of joining can create different licensing obligations (e.g., an unmodified copy will be used, a source code fragment will be copied and merged with other source code, an executable will be packaged with the distribution, a statically linked library will be used, a dynamically linked library will be used, etc.).

To achieve greater control over component re-use, teams should also take the following actions.

- Determine whether the component has been previously approved for the proposed form of use (by consulting the approved externally developed components list).

- Declare the component's version and understand its license obligations as well as those of any externally developed components or fragments it contains or depends on. This requires a declaration from the supplier of any externally developed component whose source code is unavailable for direct inspection.

- Understand all potential incompatibilities between the component's license obligations and the license obligations of other externally developed components included in this project.

- Present the above information to the project's legal, technical, and business authorizers and request approval to use the component as described.

Approvals should be reflected in the appropriate organization-wide lists.

- If the component is internal and sensitive, the list that covers these items should be updated to note that component's inclusion in this project.

- If the component is externally developed, its metadata and approval details (origin, version, license, license obligations, permitted forms of use, permitted projects, approvers, approval date) should be recorded in the list of approved external components.

Best Practice #4:
Verify every build and release

Inspecting the code base on a regular basis decreases the likelihood that unexpected components will be introduced without being noticed. Therefore, at the creation of each project build or at release assembly, the development team should verify that:

- No unapproved sensitive internally or externally developed components or fragments have been added to the project

- No unapproved changes have been made to sensitive internally developed components

- No changes have been made to externally developed components whose form of use precludes changes, or requires that all changes be approved.

Any inappropriate additions or changes discovered during verification should be immediately addressed, either by obtaining approval from the project's legal, technical, and business reviewers, or by backing out the offending modification. The root cause of any component misuse should be identified and corrected to ensure no subsequent regression.

By promptly and diligently assessing every build and release, the development team will be able to detect errors when they are least expensive to correct. At the completion of each build or release, the key metadata for all externally developed components should be recorded in the associated bill-of-materials. This enables demonstrable compliance with license obligations, and eliminates any uncertainty caused by changes between project releases by providing a clear audit trail.

Best Practice #5:
Review compliance at project phase transitions

As a project completes a major development process phase, its legal, technical, and business reviewers should do the following.

- Verify that no unapproved sensitive internal or external components or fragments are used in the project.

- Verify that no unapproved changes were made to sensitive internal components, and that no unapproved or precluded changes were made to external components.

- Review the license obligations of all external components used in the project, and ensure compliance with these obligations.

These phase reviews backstop the development team, and keep the legal, technical, and business reviewers engaged in the management of software re-use. They also verify that changes in the project's objectives have not created legal, technical, or financial inconsistencies with the licenses of components used in the project.

Best Practice #6:
Control component contribution and disposition

The rationale for contributing components to an open source project is beyond the scope of this report, as are the considerations involved in transferring ownership to a third party or creating a new open source project. However, if a contribution or transfer of a candidate component or fragment is deemed appropriate, the project's legal and business reviewers should

- determine whether the candidate component's sensitivity (if internally developed) is an impediment to contribution or transfer
- verify the right to contribute or transfer every externally developed component or fragment contained within the candidate

This helps to ensure that the organization does not inadvertently contribute code that shouldn't be contributed because of its sensitivity or because the organization is not entitled to contribute it.

Best Practice #7:
Assess software components before acquisition

If an organization is considering an acquisition that would include a significant interest in one or more software components, the designated set of legal, technical, and business reviewers should be charged with the following.

- Identifying all included components not owned by the supplier or target.
- Assessing their license obligations with respect to the acquiror's compliance, business objectives, and legal policies.
- Assessing the impact of any required rework or change on cost, revenue, and quality.

Note that this best practice applies to a variety of situations in which financial investments are involved. Such situations include: company mergers and acquisitions, product acquisitions, joint venture formations, venture capital investments, etc.

Conclusion

In summary, this report describes a set of seven best practices whose objective is to encourage development based on component re-use and software assembly. By integrating these practices in its development processes, organizations will have far greater assurances of compliance with all relevant license obligations and far more effective protection of software intellectual property.

Adopting these practices will enable companies to be more aggressive in their use of the software assembly approach. That, in turn, will enable those companies to more quickly gain the benefits and competitive advantage this new development approach promises – including accelerated project timelines, improved software quality, and reduced development costs.

We are interested in your feedback and comments on this paper. Please send them to:
 bestpractices@blackducksoftware.com.

About Black Duck Software

Black Duck Software™ is the leading provider of software compliance management solutions that help companies govern how software assets are created, managed, and licensed. Black Duck's protexIP™ suite of offerings helps businesses take maximum advantage of open source software while at the same time assure they satisfy the obligations associated with the code they use. Black Duck's customer base includes enterprises, product developers, outsourcers, law firms and other organizations worldwide that are concerned with protection of software intellectual property. For more information about Black Duck, visit www.blackducksoftware.com.

Contact

To learn more about how Black Duck Software can help your company manage license compliance and gain competitive advantages in software development, please contact sales@blackducksoftware.com or call +1.781.891.5100 x450. Additional information is available at Black Duck's website at: www.blackducksoftware.com

blackduck ™